探索アルゴリズムその2

共同実験者
075711A　山口　藍
075722F 斎藤 由利絵
075730G 沢岨　千明
055747B 中山 康弘

2008 年 12 月 30 日 火曜日

1 Level 1: 線形分離可．(2+5=7 点) 担当:055747B 中山康弘

1.1 Level 1.1

サンプルソース（bp_mo.c）は「OR 問題」を設定している．この問題においてシード値を変更して適当回数実行し，大抵の場合簡単に解けている事を確認せよ．
複数回実行した結果（ seed 値 が 10, 100, 1000, 10000 ）とそれぞれのグラフをのせる．

seed = 10

```
./a.out 10
iteration =    0, error = 0.09725
iteration =    1, error = 0.05256
iteration =    2, error = 0.06108
----------------------------------------
iteration =   93, error = 0.00010
iteration =   94, error = 0.00010
iteration =   95, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10471, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89005, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89015, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92375, t{0] = 0.9
iteration =   95, error = 0.00010
```

seed = 100

```
./a.out 100
iteration =    0, error = 0.09918
iteration =    1, error = 0.05666
iteration =    2, error = 0.06228
----------------------------------------
iteration =   96, error = 0.00010
iteration =   97, error = 0.00010
iteration =   98, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10474, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89005, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89010, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92387, t{0] = 0.9
iteration =   98, error = 0.00010
```

seed = 1000

```
./a.out 1000
iteration =    0, error = 0.07360
iteration =    1, error = 0.06211
iteration =    2, error = 0.05942
----------------------------------------
iteration =   93, error = 0.00010
iteration =   94, error = 0.00010
iteration =   95, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10473, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89000, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89018, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92380, t{0] = 0.9
iteration =   95, error = 0.00010
```

seed = 10000

```
./a.out 10000
iteration =    0, error = 0.07602
iteration =    1, error = 0.05777
iteration =    2, error = 0.05726
iteration =    3, error = 0.06371
----------------------------------------
iteration =   94, error = 0.00010
iteration =   95, error = 0.00010
iteration =   96, error = 0.00010
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10472, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.89004, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.89012, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.92382, t{0] = 0.9
iteration =   96, error = 0.00010
```

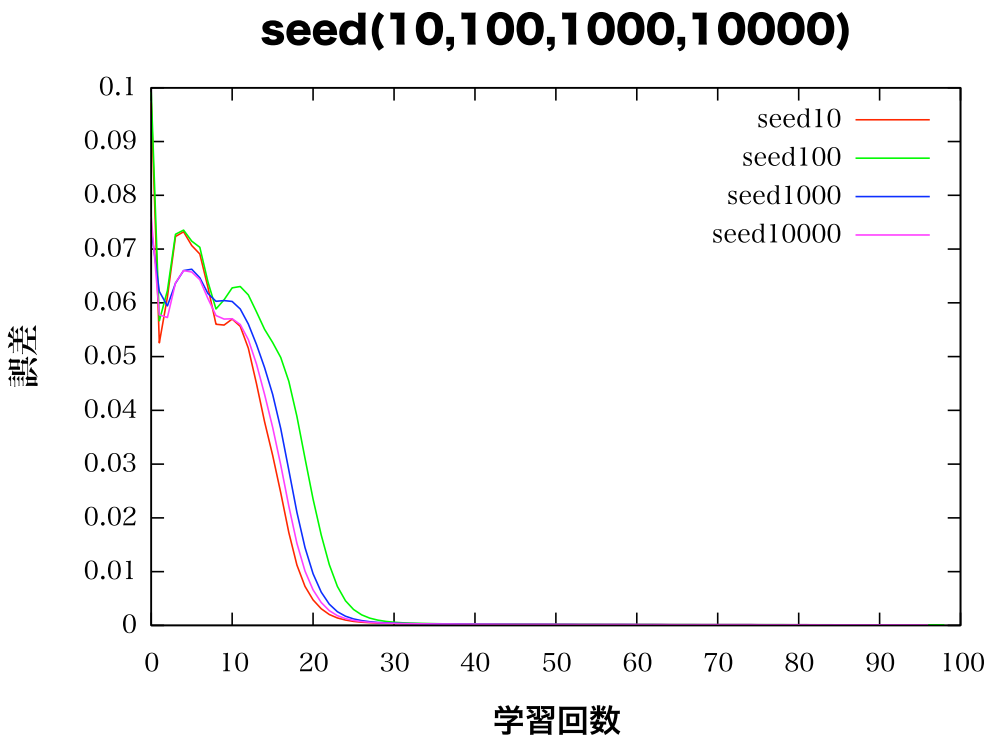

図 1: seed 値がそれぞれ、10　10000 のグラフ

1.2　考察

サンプルソースの seed 値をそれぞれ 10, 100, 1000, 10000 に設定して実行した。
記載した実行結果からも分かるように、試行回数が 100 回程度で誤差が 0.0001 以下となって終了している。
グラフでは 10 回から３０回の範囲で誤差が急速に減少している。また、この学習において 0 を 0.1、1 を 0.9 としている。これはプログラム中での演算で問題が起きないようにしている。

1.3　Level 1.2

「OR 問題」を学習させた際の学習結果をグラフ化せよ．ただし，横軸は学習回数，縦軸は誤差とする．

コメント：学習に成功（収束）した回数を Epoch（エポック）数と呼ぶ．学習結果をグラフ化する際にはエポック数を 5 とし，平均化したグラフを作成する事．shell/perl 等のスクリプト言語/表計算ソフト/gnuplot 等を利用すると楽になるでしょう．ソースファイルそのものを改変するのも OK．

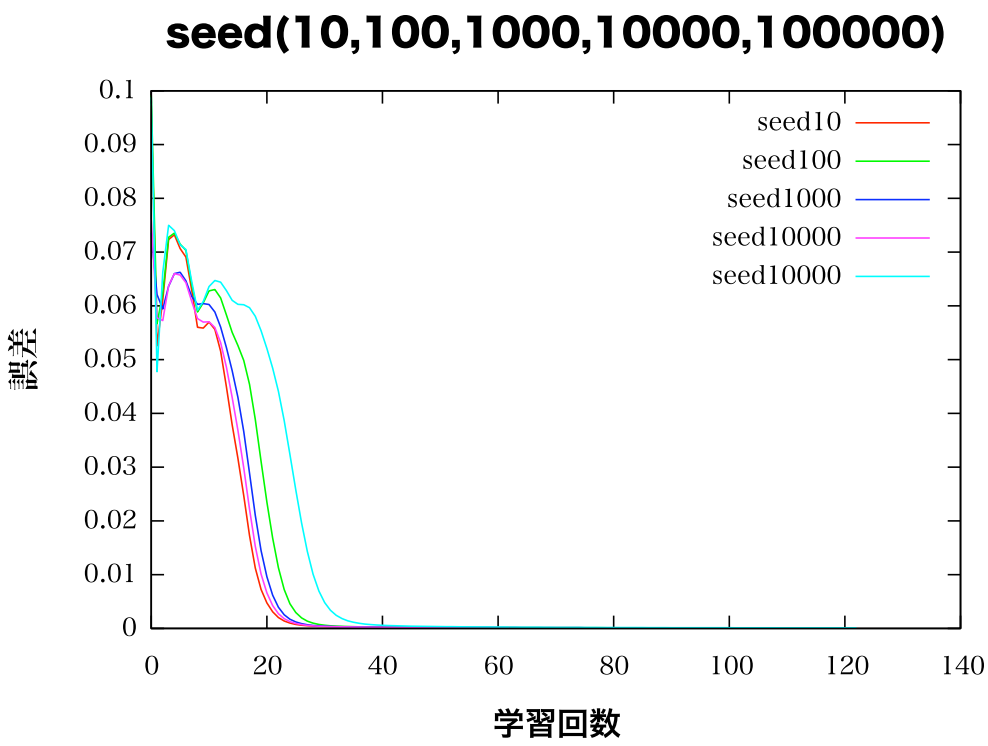

図 2: 合成前のグラフ

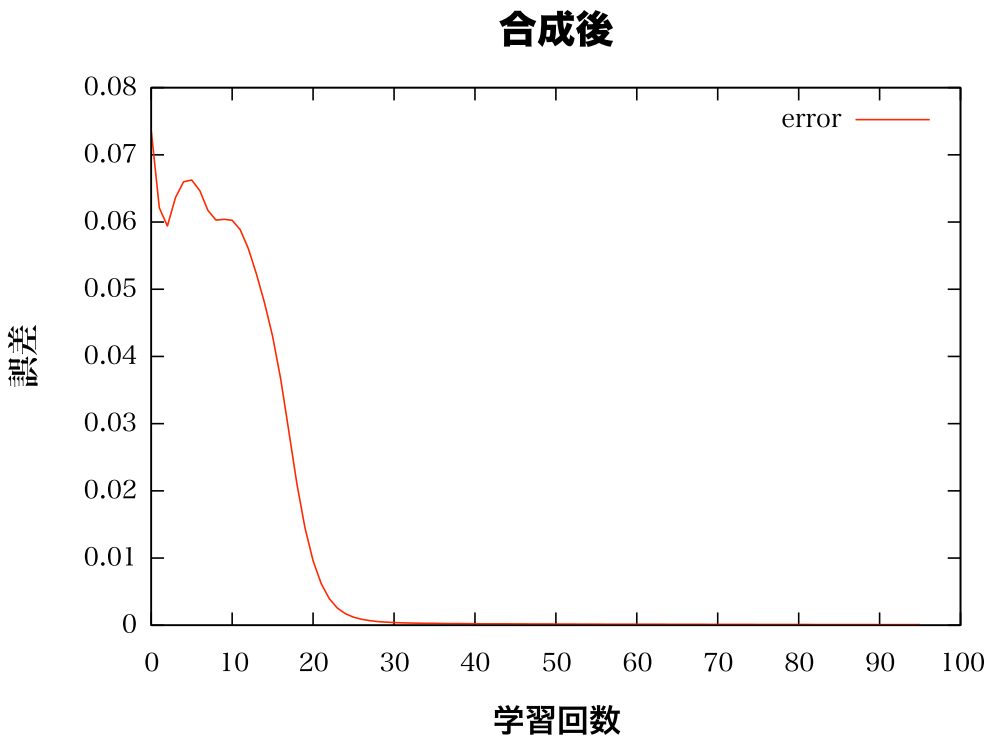

図 3: 合成後のグラフ

2 Lebel2． (5+8=13 点) 担当:075711A 山口 藍

2.1 Level 2.1

bp_mo_exor.c を編集して「ExOR 問題」へ変更し，シード値を変更して複数回実行せよ．OR 問題と異なり，学習が適切に収束しない事を確認し，何故 ExOR 問題が OR より困難なのかを説明せよ．

変更箇所
bp_mo_exor.c

```
〜〜〜〜〜〜〜〜〜〜〜〜省略〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜
  /* OR Problem */
  i_lay[0][0]=OFF; i_lay[0][1]=OFF; i_lay[0][2]=ON; teach[0][0]=OFF;
  i_lay[1][0]=ON;  i_lay[1][1]=OFF; i_lay[1][2]=ON; teach[1][0]=ON;
  //どちらかに on が入っていたら on(OR 問題)
  i_lay[2][0]=OFF; i_lay[2][1]=ON;  i_lay[2][2]=ON; teach[2][0]=ON;
  i_lay[3][0]=ON; i_lay[3][1]=ON;  i_lay[3][2]=ON; teach[3][0]=OFF;
  //teach[3][0]=ON を OFF にしたら ExOR
  〜〜〜〜〜〜〜〜〜〜〜〜省略〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜〜
```

実行結果

```
%./a.out 1000
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10036, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.64118, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.64118, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.64131, t{0] = 0.1
 100000, 0.05679
%./a.out 2000
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10036, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.64118, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.64118, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.64131, t{0] = 0.1
 100000, 0.05679
%./a.out 3000
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10036, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.64118, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.64118, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.64131, t{0] = 0.1
```

複数回シード値を変更して実行したら上記の実行結果のようになった． 実行結果から学習が適切に収束していない事がわかる．

なぜ何故 ExOR 問題が OR 問題 より困難なのかというと、OR 問題の場合は、中層ユニット数が一本で足りるのに対し、ExOR 問題は複数本必要になってくる． そのため、学習が適切に収束していなかったとなる． わかりやすく以下の図に示す．

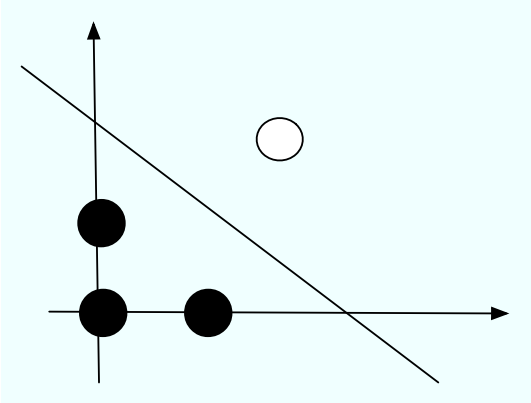

図 4: OR 問題

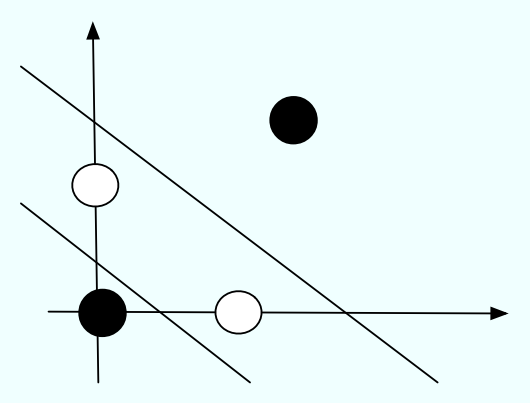

図 5: ExOR 問題

2.2 Level 2.2

「ExOR 問題」をうまく学習する為に，以下に示す各パラメータを変更して実験せよ．
最も効率良く学習が収束する組み合わせを検討せよ．
パラメータ 変更目安
学習係数 ETA 0 以上〜2.0 未満
慣性項 ALPHA 0 以上〜1.0 未満
中間層のユニット数 HIDDEN 1 以上

以下の数値が最も効率よく学習が収束する組み合わせとなる．
実行結果は以下の通りである．

パラメータ	変更目安
学習係数 ETA	1.99
慣性項 ALPHA	0.80
中間層のユニット数 HIDDEN	19

```
%./a.out 1000 | tail -n 5
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.11020, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88692, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88884, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.11696, t{0] = 0.1
iteration =  102,error =  0.00010
%./a.out 2000 | tail -n 5
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10796, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88892, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88776, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.11866, t{0] = 0.1
iteration =  107,error =  0.00010
%./a.out 3000 | tail -n 5
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10887, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88793, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88831, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.11733, t{0] = 0.1
iteration =   93,error =  0.00010
%./a.out 4000 | tail -n 5
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10500, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88973, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88880, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.12178, t{0] = 0.1
iteration =  119,error =  0.00010
%./a.out 5000 | tail -n 5
ctg[0] : i[0] = 0.1  i[1] = 0.1  o[0] = 0.10939, t{0] = 0.1
ctg[1] : i[0] = 0.9  i[1] = 0.1  o[0] = 0.88660, t{0] = 0.9
ctg[2] : i[0] = 0.1  i[1] = 0.9  o[0] = 0.88790, t{0] = 0.9
ctg[3] : i[0] = 0.9  i[1] = 0.9  o[0] = 0.11670, t{0] = 0.1
iteration =  101,error =  0.00010
```

以上の実行結果のようなシード値と収束回数になった.

3　Level 3 文字認識サンプル　　担当:075730G　澤岻　千明

ソースのあるティレクトリ (/info2/teacher/tnal/nn/nn num/) に移動、Level 3.1 〜 3.5 まて (全て必須) を実施せよ。

3.1　Level 3.1

サンプルソースを動かし、0〜9 の文字を認識する様に重みを学習させよ。本プログラムでは、学習用には learn0.txt ... learn9.txt を用いているが、類似した文字 eva1-1.txt、eva1-2.txt を適切に 1 として認識するか、確認せよ。

parameter.h の HIDDEN を 10、ITERATIONS を 10000 と変更し、実行した。

3.1.1　実行結果

ここの結果は、全て学習済みである。なお、学習させた 1 は以下のものである。

```
teach[1] = 0100000000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
```

- eva1-1.txt

```
nn> e
filename? --> ./data/eva1-1.txt
correct = 0100000000
0000010000
0000110000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
0000010000
```

```
0001111000
CHECK filename ./data/eva1-1.txt
EVA o[0] = 0.00194, correct[0] = 0.0
EVA o[1] = 0.97540, correct[1] = 1.0
EVA o[2] = 0.00030, correct[2] = 0.0
EVA o[3] = 0.02059, correct[3] = 0.0
EVA o[4] = 0.00060, correct[4] = 0.0
EVA o[5] = 0.00318, correct[5] = 0.0
EVA o[6] = 0.01224, correct[6] = 0.0
EVA o[7] = 0.00022, correct[7] = 0.0
EVA o[8] = 0.00005, correct[8] = 0.0
EVA o[9] = 0.00037, correct[9] = 0.0
EVA sum_error = 0.06408
```

EVA o[1] の値が 0.97540 となっており、最も 1 に近い。つまり、この文字は 1 であると認識されている。

- eva1-2.txt

```
nn> e
filename? --> ./data/eva1-2.txt
correct = 0100000000
0000000000
0000000000
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0000000000
CHECK filename ./data/eva1-2.txt
EVA o[0] = 0.00261, correct[0] = 0.0
EVA o[1] = 0.00126, correct[1] = 1.0
EVA o[2] = 0.00013, correct[2] = 0.0
EVA o[3] = 0.01399, correct[3] = 0.0
EVA o[4] = 0.00961, correct[4] = 0.0
EVA o[5] = 0.00266, correct[5] = 0.0
```

```
EVA o[6] = 0.00812, correct[6] = 0.0
EVA o[7] = 0.30570, correct[7] = 0.0
EVA o[8] = 0.00623, correct[8] = 0.0
EVA o[9] = 0.03465, correct[9] = 0.0
EVA sum_error = 1.38244
```

この文字では EVA o[7] の値が最も高く、0.30570 となっている。それでも、7 であると認識するには足りない。１として認識されていない。

3.2 Level 3.2

Level 3.1 において、2.2 と同様にパラメータを調整し、学習後の誤差が最も速く収束する（＝学習に向いているパラメータの）組み合わせを検討せよ。なお、エポック数５以上の平均回数での収束度合いを求めること。

パラメータ	初期設定	変更目安
学習係数 ETA	1.50	0 〜 1.99
慣性項 ALPHA	0.50	0 〜 0.99
中間層のユニット数 HIDDEN	5	1 〜 100

初期状態では平均 7500 回程行う。手動で色々試してみた結果、エポック数５以上で、ETA 1.70、ALPHA 0.55、HIDDEN 35 のパラメータが最も速く収束した。

3.3 Level 3.3

Level 3.2 で得られたパラメータによる学習曲線を出力し、パラメータと収束能力の関連性について考察せよ。

Level 3.2 のパラメータ ETA 1.70、ALPHA 0.55、HIDDEN 35 から得た学習曲線を図 6 にしめす。

図からみても分かる様に、始めの方で一気に 0 に近づいている。iteration の 30 付近から 0 に近くなるため、どのようになっているのかがわからない。そこで、この図 6 の一部分を次の図 7 に示す。

パラメータ、つまり ETA、ALPHA、HIDDEN の値が大きい程 性能が良くなる傾向にある。だが、単純に大きいからといって最大値が最良の結果を出す訳ではない。

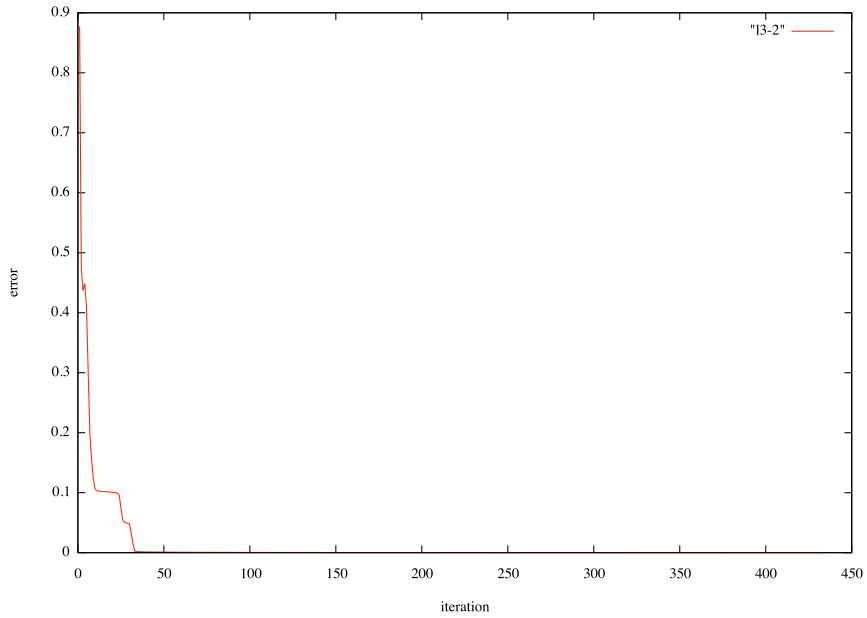

図 6: Level3.2 から得たパラメータの学習曲線

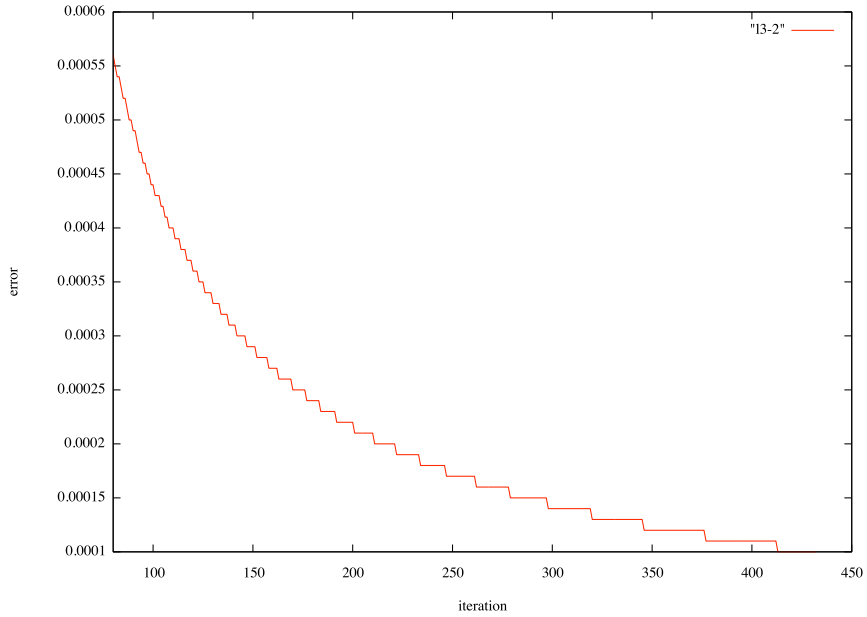

図 7: 曲線の細部

3.4 Level3.4． 担当:075722F 斉藤 由利絵

各自で任意の評価用データを複数作成し，学習時のデータとの違いが少ないほど認識率が高い事を示せ．
図 8 のように徐々に元の「learn0.txt」の状態にする．

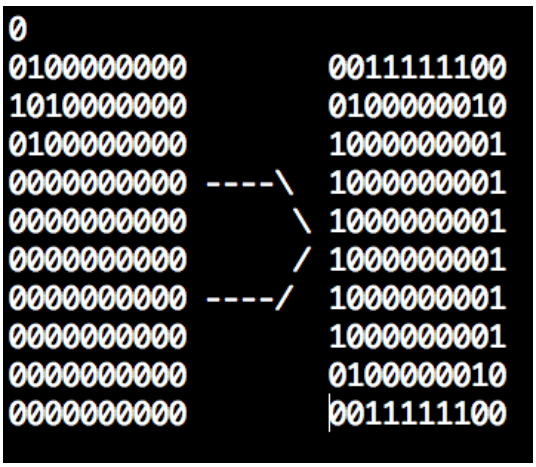

図 8: learn0.txt

	認識率
初期状態	0.00005
	0.00079
	0.00026
	0.01180
	0.07656
	0.94956
元の状態	0.98842

表 1: 認識率表

3.5 考察

この結果から，元のデータとかさなる部分が多いほど認識率が高くなる事が分かる．
ただし，一部認識率が減っている部分があるが，これは元のデータと同じようにするため一部へこませた所があるので，そこが原因だと考えられる．

3.6 Level3.5

下図のように入力されたデータが想定していた入力と比べてサイズが異なったり，
位置がずれている等，文字の一部がかけている以外にも様々なデータの欠落が考えられる．
認識率を高めるにはどのような点を工夫すればよいかどのような方法でもかまわないので検討せよ．

3.7 解決策

学習データをいくつか用意し，どの位置においてもその数字を認識させるようにする方法.
位置の違いや大きさの違い等も，このデータを利用し認識させるようにする.
もし，データが今までのデータにない大きさ位置だった場合も若干のずれであれば認識する事ができる.
もし，今までのデータを大きく外れた場合，その数字をその数字だと修正認識するプログラムが
あればそのデータも追加され学習するたびにプログラムも賢くなる事ができる.

4　Level 4: NNの応用例を検討　　(4+6=10点) 担当:075715C 嘉陽　裕香

手書き文字認識の様に，(通常の）コンピュータの持つ特徴と異なり，曖昧な処理（認識・予測・類推・連想・分析・解析 … ）が得意である一方，厳密な処理は苦手である．

4.1　文字認識

問題の解説

パターン認識の種類に文字認識がある，手書き文字の認識を応用させる例の一つとして，DS の漢字検定というソフトが有る．漢字検定の問題が表示されて，その答えを手書きで書いて正誤を問う問題である．

入力層 (データの入れ方）

漢字検定の問題が表示され，その答えをペン（もしくは指）を使って，ディスプレイのタッチ画面に書いていく．書き終えたら，少し経つと認識してくれる．

出力層（出力方法）

ディスプレイに認識した文字が表示される．

ニューロン数

コンピュータが扱える形式になっているので，0 と 1 の２種類となるから，ニューロン数は２つ必要である．

4.2　画像認識

問題の解説

顔，眼底，瞳，指紋，掌紋等

入力層 (データの入れ方）

カメラによる撮影

出力層（出力方法）

これも文字認識と同じく，ディスプレイに結果が表示される．印刷する事によって紙媒体でも出力する事が出来る．

ニューロン数

これも文字認識と同じく，ニューロン数は２つになる．

5　Level X: 補足　　担当:全員まぜこぜ

5.1　実験の内容・進め方に関するコメント等

1. 進め方に関しては，とても良いと思う．レポートの開示に関しては賛成です．制限を付けずにそのままにしておいたら，これからの後輩達が楽できそうだと思います．
2. やって欲しかった内容に関しては，授業でやっている事が，何処で活躍しているのか，一番最新の物の紹介です．個人的な意見になりますが，一番最近はどこで活躍しているかというのが明確になっていれば良いと思います．

つかれた．授業中にもっと具体的なところまで見たかった．

参考文献・引用文献

Top - HW graveyard
http://www.ie.u-ryukyu.ac.jp:16080/ e065762/wiki/
漢検 DS
http://www.rocketcompany.co.jp/kanken/
level 3.1 〜 3 参考文献 :
http://133.13.48.8/~e065762/reports/exp2/ex2_7.pdf